Package Delivery System 2.0

20181625 남현준

1. BCNF decomposition에 대한 설명

A. BCNF decomposition

i. Customer Entity

Customer entity에서 있을 수 있는 functional dependency는 다음과 같다.

1. Security number -> Name
2. Security number -> Phone number

다음 functional dependency들에서 얻을 수 있는 closure는 다음과 같다.

- (Security number)$^+$ = {security number, name, phone number}

해당 functional dependency들은 BCNF를 만족하고 있다. Functional dependency의 left part에 해당되는 부분이 customer entity의 superkey인 security number이고 모든 non-candidate인 attribute들이 candidate key에 대해 종속성을 가지기 때문이다.

ii. Account entity

Account entity에서 있을 수 있는 functional dependency는 다음과 같다.

1. Account ID -> Account name
2. Account ID -> security number
3. Security number -> Account ID

해당 entity가 customer와 형성하는 relationship에서, 1인 1계정을 원칙으로 한다고 가정했기 때문에, security number로 account ID를 유일하게 식별할 수 있고 따라서 security number -> account ID인 functional dependency가 존재하게 된다.

위에서 구한 functional dependency들의 closure를 구하게 되면

- (account ID)$^+$ = {account ID, account name, security number},
- (security number)$^+$ = {account ID, account name, security number}

이다. BCNF simplified test를 통해 존재하는 functional dependency들이 BCNF의 조건을 위반하지 않는 것을 확인할 수 있고 따라서, 해당 entity는 decomposition을 통해 분해될 필요가 없다

iii. Payment entity

Payment entity에서 존재하는 functional dependency를 나타내면 다음과 같다.

1. Payment ID -> payment date
2. Payment ID -> Payment amount
3. Payment ID -> Payment method
4. Payment ID -> Payment status
5. Payment ID -> Account ID

위에서 구한 functional dependency들의 closure를 구하게 되면

- (payment ID)$^+$ = {payment ID, payment date, payment amount, payment method, payment status, account ID}

이다. 해당 functional dependency에서, BCNF의 조건에 위배되는 functional dependency가 존재하지 않는다. Payment ID의 closure는 해당 entity의 모든 attribute들을 포함하고 있기 때문 payment ID는 해당 entity의 superkey이다. 따라서, payment entity는 BCNF를 만족하고 있다고 할 수 있다.

iv. Package entity

Package entity에서 있을 수 있는 functional dependency는 다음과 같다.

1. Package ID -> Security number
2. Package ID -> Shipment ID
3. Package ID -> Recipient ID
4. Type of package, weight -> Price
5. Payment ID -> Package ID

위에서 구한 functional dependency들의 closure를 구하게 되면

- (Package ID)$^+$ = { Type of package, Weight, Price, Security number, Shipment ID, Recipient ID },
- (Type of package, weight)$^+$ = { Type of package, weight, price },
- (payment ID)$^+$ = { Type of package, Weight, Price, Security number, Shipment ID, Recipient ID }

이다. 해당 functional dependency들 중, type of package, weight -> price의 closure가 relation의 모든 entity들을 포함하지 않기 때문에 BCNF의 조건 중 하나인 left part가 superkey일 조건을 위배하게 된다. 따라서, 해당 functional dependency에 포함되어 있는 attribute들은 BCNF decomposition rule을 통해 분해할 필요가 있다.

일반적으로 택배를 보내게 될 때, 보내는 택배의 종류와 택배의 무게에 따라서 배송비가 책정되게 된다. 택배의 종류에 따라서 일정 범위 내에 있다면, 해당 분류에 맞춰서 가격을 책정하는 형식인 것이다. 따라서, package entity에서 type of package와 weight를 알 때, 그 때의 package price를 유일하게 알 수 있을 것이다. 하지만, 여기서 문제가 되는 부분은 Type of package, weight -> Price인 functional dependency가 hold되지만 type of package와 weight가 entity의 superkey가 아니라는 것이다.

해당 functional dependency는 BCNF의 조건에서, a -> B인 functional dependency가 존재할 때, a 부분은 superkey여야 하는 조건을 위배하고 있다. Package entity를 BCNF form으로 만들기 위해서는 entity를 decompose할 필요가 있다.

따라서, BCNF decomposition rule인 (a U B)와 (R – (B – a))을 반영하여 R1 = {package ID, type of package, weight, security number, shipment ID, recipient ID, payment ID}, R2 = {type of package, weight, price}로 package entity를 분해할 수 있을 것이다.

v. Recipient entity

Recipient entity에서 있을 수 있는 functional dependency는 다음과 같다.

1. Recipient -> Address

해당 functional dependency는 BCNF를 만족하고 있다. Functional dependency의 left part에 해당되는 부분이 recipient entity의 superkey인 security number이고 모든 non-candidate인 attribute들이 candidate key에 대해 종속성을 가지기 때문이다.

vi. Information entity

Information entity에서 있을 수 있는 functional dependency는 다음과 같다.

1. Package ID -> Type of product, product name, purpose of shipment

위에서 구한 functional dependency들의 closure를 구하게 되면

- $(\text{package ID})^+$ = {package ID, Type of product, product name, purpose of shipment },

이다. Package ID의 closure는 해당 entity의 모든 attribute를 포함하고 있기 때문에 superkey이다. 따라서, functional dependency의 left part에 해당되는 부분이 recipient entity의 superkey인 security number이고 모든 non-candidate인 attribute들이 candidate key에 대해 종속성을 가지고 있으므로 해당 entity는 BCNF의 조건을 만족하고 있다.

vii. Shipment entity

Recipient entity에서 있을 수 있는 functional dependency는 다음과 같다.

1. Shipment ID -> Manager ID

위에서 구한 functional dependency들의 closure를 구하게 되면

- $(\text{shipment ID})^+$ = {shipment ID, Manager ID },

이다. Shipment ID의 closure가 entity의 모든 attribute를 포함하고 있기 때문에 shipment ID 는 superkey이다. 따라서, functional dependency의 left part에 해당되는 부분이 recipient entity의 superkey인 security number이고 모든 non-candidate인 attribute들이 candidate key에 대해 종속성을 가지기 때문에 해당 entity는 BCNF 조건을 만족하고 있다.

viii. Delivery entity

Delivery entity에 존재할만한 functional dependency는 다음과 같다.

1. Tracking number -> Picked up time/date, Estimate delivery time/date, Timeliness of delivery, Actual arrival time, Shipment ID, Transportation ID, Transportation status, location ID
2. Shipment ID -> picked up time/date, estimated delivery time/date, timeliness of delivery, actual arrival time/date

위에서 구한 functional dependency들의 closure를 구하게 되면

- (Tracking number)$^+$ = { Picked up time/date, Estimate delivery time/date, Timeliness of delivery, Delivery status, Actual arrival time, Shipment ID, Transportation ID, Transportation status, location ID},
- (Shipment ID)$^+$ = { Tracking number, picked up time/date, estimated delivery time/date, timeliness of delivery, actual arrival time/date },

이다. 여기서, shipment ID의 closure가 relation의 모든 attribute를 포함하지 않는 것을 확인할 수 있다. 이는, 해당 functional dependency가 BCNF 조건 중 하나는 left part가 superkey일 조건을 위반하고 있다는 의미이다. 따라서, 해당 functional dependency는 decomposition rule을 통해 분해될 필요가 있다.

Rule for decomposing schemas not in BCNF에 따르면 decomposed schema는 (a U B)와 (R – (B – a))로 이루어지게 된다. 여기서 (a U B)는 {shipment ID, picked up time/date, estimated delivery time/date, timeliness of delivery, actual arrival time/date}, (R – (B – a))는 {tracking number, delivery status, shipment ID}가 된다.

따라서, 분해된 schema는 R1 = {shipment ID, picked up time/date, estimated delivery time/date, timeliness of delivery, actual arrival time/date}, R2 = {tracking number,

delivery status, shipment ID}가 되며, 분해된 schema R1과 R2는 BCNF를 만족하게 된다.

ix. Location entity

Location entity에서 있을 수 있는 functional dependency는 다음과 같다.

1. Location ID -> Location name, Address

위에서 구한 functional dependency들의 closure를 구하게 되면

- (Location ID)$^+$ = { Location ID, Location name, Address},

이다. Location ID의 closure는 entity의 모든 attribute를 포함하고 있다. 따라서, location ID는 해당 entity의 superkey이다. 즉, entity의 모든 non-candidate attribute들은 location ID에 종속성을 가지게 된다. 즉, 해당 entity는 BCNF의 조건에 위배되는 부분이 없기 때문에 BCNF를 만족하고 있다고 할 수 있다.

x. Transportation entity

Transportation entity에서 있을 수 있는 functional dependency는 다음과 같다.

1. Transportation ID, Type of transportation -> Capacity, transportation status, Shipment ID

위에서 구한 functional dependency들의 closure를 구하게 되면

- (Transportation ID, Type of transportation)$^+$ = { Transportation ID, Type of transportation, Capacity, transportation status, Shipment ID },

Transportation ID, Type of transportation의 closure는 entity의 모든 attribute를 포함하고 있다. 따라서, BCNF의 rule에서 functional dependency의 left part가 schema의 superkey일 것이라는 조건을 만족하고 있고 functional dependency가 trivial하지 않았기 때문에 BCNF의 조건을 만족하게 된다.

따라서, 해당 entity는 BCNF를 만족하고 있으며 decomposition을 통해 분해를

진행하지 않아도 된다고 판단했다.

xi. Shipment manager entity

Manager entity에서 있을 수 있는 functional dependency는 다음과 같다.

1. Manager ID -> Name, Affiliation, Phone number

위에서 구한 functional dependency들의 closure를 구하게 되면

- $(\text{Manager ID})^+$ = { Manager ID, Name, Affiliation, Phone number },

Manager ID의 closure는 entity의 모든 attribute를 포함하고 있다. 따라서, BCNF의 rule에서 functional dependency의 left part가 schema의 superkey일 것이라는 조건을 만족하고 있고 functional dependency가 trivial하지 않았기 때문에 BCNF의 조건을 만족하게 된다.

따라서, 해당 entity는 BCNF를 만족하고 있으며 decomposition을 통해 분해를 진행하지 않아도 된다고 판단했다.

2. Decomposition으로 인해 새롭게 형성되는 relationship 및 기존 entity의 변경점

A. Package entity

Package entity에서는 Type of package, weight -> Price와 Payment ID -> Package ID인 functional dependency들이 있었기 때문에 decomposition을 진행했다. Decomposition을 통해 얻은 새로운 entity들은 다음과 같다.

- R1 = {package ID, type of package, weight, security number, shipment ID, recipient ID}
- R2 = {type of package, weight, price}

새롭게 생성된 R1은 기존에 있던 relationship 중에서 customer, recipient, shipment와의 relationship을 유지할 수 있도록 연결해줄 것이다. 해당 entity는

package의 정보를 저장하고 있기 때문에 해당 relationship들을 유지해야 초기 설계의 방향성을 유지할 수 있기 때문이다.

R2는 새롭게 정의되는 entity로 일정 무게범위 이내의 특정 package에 대한 값을 저장하게 될 것이다. R2를 R1과 relationship을 형성하게 하여 해당 package의 price에 대한 정보도 유지할 수 있게 할 것이다.

B. Delivery entity

BCNF decomposition을 통해 다음과 같은 relation들로 분해를 진행했다.

- R1 = {shipment ID, picked up time/date, estimated delivery time/date, timeliness of delivery, actual arrival time/date},
- R2 = {tracking number, delivery status, shipment ID, location ID}

R1을 shipment_time, R2를 shipment_delivery라고 하겠다.

Shipment_time은 특정 shipment의 시간에 관련되어 있는 정보를 저장하게 될 것이다. Shipment_delivery는 shipment의 delivery와 관련된 정보를 저장하게 될 것이다. 기존에 delivery entity가 shipment, location과 형성하던 relationship에서 shipment_time, shipment_delivery 모두 shipment entity와 새롭게 relationship을 형성하도록 할 것이다. Location의 경우, 택배의 배송을 조회하면서 같이 나오는 것이 일반적이기 때문에, shipment_delivery와 location을 연결하여 기존의 relationship을 비슷하게 유지할 수 있도록 할 것이다.

C. Transportation entity

기존의 transportation entity의 primary key는 해당 transportation의 식별할 수 있는 고유의 식별번호를 primary key로 설정한다고 했었다. 하지만, 실제로 mysql 등을 이용해 db를 구현해본 결과, 동일한 transportation이 delivery를 진행했다는 것을 저장함에 있어 error message를 받게 되었다. 따라서, 해당 entity의 attribute에 수정이 필요하다는 점을 알 수 있었다.

해당 의도하지 않은 error를 막기 위해서는 primary key에 변화가 필요하다. 일반적으로, transportation은 delivery를 진행하고 있을 때 active, 진행하지 않고 시동이 꺼진 상태라면 inactive 상태이다. 따라서, 한 식별번호 당 2개의 state가 존재할 수 있게 되는데 기존의 primary key인 식별번호 하나만으로는 이러한 정보를 저장할 수 없었다. 따라서, primary key에 변화를 주어야겠다는 판단을 내렸다.

새롭게 정의되는 primary key는 transportation_ID, transportation_status, shipment_id가 될 것이다. 이렇게, 두 개의 attribute로 primary key를 정의하게 되면, 하나의 식별번호 당 2개의 state를 저장할 수 있게 된다.

또한, 이렇게 바꿀 경우, shipment와 형성하는 relationship에도 변화가 생겨야 한다. shipment에서 foreign key로 transportation entity에 해당 transportation의 정보가 있는지 확인할 수 있어야 한다. 한 화물에 다른 shipment들이 배정될 수 있기 때문에, shipment_ID를 transportation entity의 primary key중 하나로 설정하여 한 화물이 여러 shipment와 pair를 이룰 수 있도록 구현했다.

3. Physical schema와 ODBC implementation에 대한 설명

A. Physical schema diagram에 대한 설명

Physical schema diagram에서는 logical schema diagram을 진행하면서 설정했던 entity들과 해당 entity들의 attribute, attribute 속성이 같이 출력된다. 특정 entity의 attribute 정보만 알 수 있던 logical schema diagram과는 달리, physical schema diagram에서는 schema를 구성할 때 사용한 attribute의 data 정보도 알 수 있게 되어 더욱 직관적으로 해당 schema diagram에 관한 정보를 전반적으로 알 수 있게 되었다.

i. Customer entity

Customer entity는 길이가 13인 string을 정보로 가지는 security_number attribute를 primary key로 가진다. 해당 entity의 attribute로는 name, phone_number가 있으며 각각 varchar(18), char(11)로 선언되어 해당 타입의 데이터를 입력으로 받게 된다.

해당 entity에서 security_number, name은 not null 설정을 주었다. Security_number는 특정 사람을 식별하기 위해 존재하는 값이고, 이름의 경우에도 모든 사람이 기본적으로 이름을 가지고 있을 것이기 때문에 null을 허용하지 않았다. Phone_number의 경우, 흔치는 않지만 간혹 전화를 사용하지 않는 사람도 있을 수 있기 때문에 null을 허용하도록 하였다.

Security_number, name, phone_number은 문자열을 저장하기 때문에 domain을 string으로 설정하였다.

Customer entity는 account, package, shipment manager entity와 관계를 형성하게 된다. Customer_account relationship에서 구현 상으로 하나의 고객이 하나의 계정만을 생성할 수 있다고 가정했기 때문에 customer는 account와 1:1의 relationship을 가지게 된다.

Customer_package relationship에서 customer entity는 1:N의 relationship을 가진ㄴ다. 서비스를 이용하게 될 때, 한 명의 이용자가 서비스를 여러 번 사용해서 물품을 여러 번 회사에 맡기는 일이 있을 수도 있다. 따라서 해당 relationship에서 customer는 one-to-many의 관계를 형성하게 될 것이고, 이에 따라 relationship cardinality 또한 1:N이 될 것이다.

Shipment manager와의 relationship에서는 한 명의 customer가 여러 shipment manager와 계약을 맺을 수 있고, 마찬가지로 한 명의 shipment manager도 여러 customer와 계약을 맺을 수 있다. 따라서, 해당 관계는 many-to-many이며 N:N으로 표기될 것이다. Physical schema에서는 해당 관계가 분해되어 customer_manager라는 relation이 추가적으로 생성되게 되었는데 이 relation의 주요 목적은 customer와 shipment_manger 간의 relationship을 형성하여 그와 관련된 정보를 얻을 수 있게 하는 것이다.

ii. Account entity

Account entity는 최대 길이 18인 string을 정보로 가지는 account_ID를 primary key로 가지게 된다. 해당 entity의 attribute인 account_name는 최대 길이가 18인 string을 정보로 가지게 된다. Security_number는 customer entity를 referencing하는 foreign key이고 customer에 선언된 security_number와 같은 데이터형을 가지게 된다.

해당 entity에서, 모든 attribute들은 null 값을 허용하지 않도록 설정했다. Primary key의 경우는 원래 null값이 존재할 수 없고, account_name의 경우 서비스를 사용할 때 필연적으로 해당 서비스에서 사용할 닉네임 (닉네임을 사용하지 않더라도 실명)이 account의 정보와 연동될 것이기 때문에 null을 허용하지 않았다. Security_number는 customer entity를 referencing하는 foreign key이고 이를 확인하기 위해서 customer에 해당 값이 존재하는지 알 수 있어야 한다. 따라서, 해당 attribute 또한 마찬가지로 null을 허용하지 않아 무결성 체크를 할 수 있게 설정했다.

Account_ID, account_name, security_number는 문자열을 저장하기 때문에 domain을 string으로 설정했다.

Payment entity와는 1:N의 관계를 형성하게 된다. 한 사람이 서비스를 여러 번 이용하여 물품을 여러 사람에게 보내거나 한 사람에게 여러 번 보낸다는 가능성이 있기 때문이다. 앞서 말한 것과 같이 Customer_account relationship에서 구현 상으로 하나의 고객이 하나의 계정만을 생성할 수 있다고 가정했기 때문에 customer는 account와 1:1의 relationship을 가지게 된다.

iii. Payment entity

Payment entity는 integer형의 데이터를 가지는 payment_ID가 primary key로 설정되어 있다. 해당 entity의 attribute들인 payment_amount, payment_date, payment_method, payment_status들은 각각 integer, datetime, varchar(18), varchar(18)로 선언되어 있다. 이 때, 한 가지 logical schema에서의 설정과 다른 부분이 있다. Payment_date의 데이터형은 logical schema에서 DATETIME DAY TO MINUTE 였지만, physical schema에서는 datetime으로 설정되었다.

Payment의 모든 attribute는 not null로 설정되었다. 해당 entity는 사용자가 서비스를 이용했을 때의 결제내역을 저장하게 된다. 이 때, 처리가 진행되었거나, 진행 중이거나 아직 진행되지 않은 상태 중 하나의 상태를 status로 가지고, 그 때의 지불해야 할 금액, 지불을 시도하는 날짜 (혹은 진행된 날짜) 등의 정보는 모두 유의미하기 때문에 필수적이다. Account_ID의 경우 foreign key이고 해당 값을 통해 account에 존재하는 account에 대한 payment인지를 확인할 수 있기 때문에 not null이 되어야 한다.

Payment_ID, payment_amount는 정수를 저장하기 때문에 number, payment_date는 시간/날짜를 저장하기 때문에 datetime, 그 외 나머지 attribute들은 문자열을 저장하기 때문에 string으로 domain을 설정했다.

Account entity와는 1:N의 관계를 형성하게 된다. 한 사람이 서비스를 여러 번 이용하여 물품을 여러 사람에게 보내거나 한 사람에게 여러 번 보낸다는 가능성이 있기 때문이다.

Package와는 1:1 relationship을 형성하게 될 것이다. 하나의 지불정보에 대해 한 package가 관련되어 있을 것이며 마찬가지로 한 package도 하나의 지불 정보만을 가지게 될 것이기 때문이다.

iv. Package entity

Package entity는 integer형의 데이터를 가지는 package_ID가 primary key로 설정되어 있다. 해당 entity의 나머지 attribute들은 모두 foreign key로 각각 연결되어 있는 relation들을 referencing 하여 무결성 체크를 진행하게 될 것이다.

해당 entity의 모든 attribute는 null값을 가질 수 없게 설정했다. Primay key인 package_ID는 entity의 tuple들을 유일하게 식별하기 위해 not null로 설정했고, 나머지 attribute들은 모두 foreign key이므로 해당 값을 통해 무결성 체크를 진행하여야 한다. 하지만, foreign key에 null이 들어올 경우, 무결성 체크에 애로사항이 생기기 때문에 not null로 설정해주었다.

Package_ID, weight, shipment_ID, payment_ID는 숫자와 관련된 정보를 저장하기 때문에 number, 그 외 나머지 attribute들은 문자열을 저장하기 때문에 string으로 domain을 설정했다.

customer와 package 사이의 관계는 package가 many, customer가 one인 many-to-one의 관계가 되고, 이를 나타내면 N:1이 된다.

shipment는 출발지와 목적지가 같은 package들을 한 곳에 모아서 출발하는 package의 묶음이다. 따라서, shipment에는 목적지가 같은 package들이 있을 것이고이를 cardinality로 나타내면 package가 many, shipment가 one, 즉 1:N(혹은 N:1)로 나타낼 수 있다.

payment와는 1:1 relationship을 형성하게 될 것이다. 하나의 지불정보에 대해 한 package가 관련되어 있을 것이며 마찬가지로 한 package도 하나의 지불 정보만을 가지게 될 것이기 때문이다.

v. Recipient entity

Recipient entity는 물품을 받게 되는 사람의 간단한 정보를 저장하는 entity이다. Primary key로 recipient_ID가 char(13)형으로 선언되어 있고 최대 길이 30글자까지 저장될 수 있는 varchar(30)형으로 address attribute가 정의되어 있다.

Recipient_ID와 address 모두 not null로 설정되어 있다. Recipient_ID의 경우 tuple을 unique하게 식별하기 위해 null값이 들어올 수 없고, address의 경우, 추후에 query에서 join을 통해 유의미한 정보를 얻어야 할 수 있기 때문에 null 값을 허용하지 않도록 설정했다.

해당 entity의 attribute들의 domain을 string으로 설정했다.

Recipient는 package와 relationship을 형성하게 된다. 한 recipient가 배송을 여러 번 받을 수 있기 때문에 해당 관계는 N:1이 된다.

vi. Package_price

Package_price는 package의 종류와 무게에 따라 산출된 그 때의 가격을 저장하게 되는 entity이다. Primary key로 type_of_package와 weight를 가지고 있으며 각각 varchar(18), integer 자료형으로 선언되어 있다. 해당 entity의 attribute인 price는 integer 자료형으로 선언되어 종류와 무게에 따라 다른 가격의 정보를 저장하게 될 것이다.

Package_price의 모든 attribute는 not null로 설정되었다. 이런 판단을 한 이유는 type_of_package와 weight는 해당 entity의 값들을 unique하게 식별할 때 사용되는 key 이기 때문에 null 값이 들어올 수 없다. Price의 경우, 배송되는 모든 품목에 대해 해당 품목을 배송하기 위해 책정되는 가격이 있을 것이고, 이를 활용하여 query 등을 통해 정보를 얻을 때 사용될 수 있다. 따라서, null이 있을 경우 해당 작동을 할 때 애로사항이 있게 될 것이므로 not null로 설정한다는 결론을 내렸다.

Type_of_weight는 string, 나머지 attribute들은 number로 domain을 설정했다.

Package_price는 package와 relationship을 형성한다. 이 relationship에서 package가 특정 조건일 때, 고정된 값이 price로 책정되게 된다. 따라서, 해당 관계는 하나의 package type, weight에 대해 여러 package가 할당될 수 있고 이는 N:1의 relationship임을 의미한다.

vii. Information

Information entity는 package의 부가적인 정보를 저장하게 되는 entity이다. 해당 entity의 primary key인 package_ID가 foreign key로 설정되어 있는 것을 확인할 수 있을 것이다. 이는, information entity가 package에 종속된 weak entity이고 package의 정보가 존재할 때에만 information entity의 정보가 유효하다는 것을 physical schema diagram 상에서 도출할 수 있게 해준다. 해당 attribute의 자료형은 integer로 referencing하는 package의 package_ID의 자료형과 일치한다. 나머지 attribute들인 type_of_product, product_name, purpose_of_shipment는 모두 최대 글자수가 18만큼 저장할 수 있는 varchar(18) 자료형으로 선언되어 있는 것을 확인할 수 있다.

해당 entity는 primary key인 package_ID not null로 설정되어 있다. Package_ID의 경우 primary key이자 foreign key이고 이를 통해 entity의 tuple들을 unique하게 식별 및 무결성 체크를 하기 때문에 null 값이 허용될 경우 database의 신뢰도가 떨어지게 될 것이다. 다른 attribute들의 경우, 기본적으로 저장하지 않아도 되는 부가적인 데이터들을 저장하기 때문에, null을 허용해도 무방하다는 판단을 내렸다.

Foreign key인 package_ID를 제외하고 나머지 attribute들의 domain은 string으로 설정했다.

Weak entity인 경우, 관계되어 있는 entity의 cardinality를 따라가게 된다. 이 경우, information entity는 1:N의 cardinality를 가지게 될 것이다.

viii. Shipment entity

Shipment는 package를 운송하게 되는 shipment에 대한 정보를 저장한다. Shipment_ID가 primary key로 설정되어 있으며 integer 자료형을 가지고 있다. 해당 entity의 attribute인 manager_ID는 manager entity를 referencing하는 foreign key이며 integer 자료형을 가지게 된다.

Shipment_ID와 manager_ID는 null 값을 허용하지 않도록 설정했다. Shipment_ID는 primary key로 해당 entity의 tuple들을 unique하게 식별할 수 있어야 하기 때문에 null이 들어올 수 없다. Manager_ID의 경우 manager entity를 referencing하며 해당 값을 통해 무결성 체크를 하기 때문에 null이 들어올 경우, manager entity에 없는 경우에도 문제가 없다는 결론이 나올 수 있기 때문에 null을 허용하지 않았다.

해당 entity의 attribute들의 domain은 number로 설정되었다.

Shipment_delivery와는 1:1 relationship을 형성한다. 하나의 shipment에 대해 하나의 배송 정보만이 존재할 수 있기 때문이다.

마찬가지로, shipment_time과도 1:1 relationship을 형성한다. 하나의 shipment에 대해서 하나의 배송시간에 대한 정보를 가진 dataset이 존재할 것이기 때문이다.

한 개의 shipment에 대해서, 배정될 수 있는 transportation의 종류는 여러 가지이다. 따라서, shipment와 transportation 사이의 relationship 1:N이 될 것이다.

한 명의 shipment_manager가 여러 shipment를 담당할 수 있기 때문에 해당 관계에서는 1:N의 relationship이 생기게 된다.

ix. Shipment_delivery entity

Shipment_delivery는 shipment의 배송 현황 및 현재 위치 정보를 저장하게 되는 entity이다. Primary key로 shipment를 고유하게 식별할 수 있게 tracking_number를 가지게 되며 varchar(18)의 자료형으로 최대 18글자까지 정보에 저장할 수 있다. Delivery_status는 현재의 배송 현황을 저장하게 될 것이며 이 또한 마찬가지로 varchar(18)로 선언되어 있다. Foreign key로는 shipment_ID와 location_ID를 가지게 되며 각각 shipment와 location entity를 referencing 하여 무결성 체크를 하게 된다.

Shipment_ID는 number, timeliness_of_delivery는 string, 그 외 나머지 attribute들의 domain은 datetime으로 설정되었다.

해당 entity의 모든 값을 null을 허용하지 않도록 설정했다. Primary key는 unique하게 tuple들을 식별할 수 있어야 하기 때문에 null을 허용하지 않도록 했다. Delivery_status의 경우 모든 delivery가 모두 어떠한 status를 가지게 되는 것이 경험적으로나 구현적으로 맞다는 판단이 들었기 때문에 null을 허용하지 않았다. Shipment_ID

와 location_ID는 foreign key이고 해당 값을 통해 해당 값을 통해 무결성 체크를 하기 때문에 null이 들어올 경우, 해당 entity들에 존재하지 않는 경우에도 문제가 없다는 결론이 나올 수 있기 때문에 null을 허용하지 않았다.

shipment와는 1:1 relationship을 형성한다. 하나의 shipment에 대해 하나의 배송 정보만이 존재할 수 있기 때문이다.

location과는 1:N의 relationship을 형성한다. 배송 정보를 조회하게 될 때, location의 경우 계속 바뀔 수 있기 때문에 하나의 배송 정보가 여러 location에 대응될 수 있어야 한다.

Shipment_time과는 1:1 relatioship을 형성하게 될 것이다. 하나의 배송 정보에는 하나의 시간 정보만 있을 것이기 때문이다.

x. Shipment_time entity

Shipment_time entity는 shipment_ID를 primary key로 가진다. 즉, 해당 정보는 shipment를 referencing 하는 동시에 해당 entity들에 값이 존재해야만 존재할 수 있는 weak entity임을 암시하고 있다. 해당 entity의 attribute인 picked_up_time_date, estimated_delivery_time_date, timeliness_of_delivery, actual_arrival_time_date는 각각 datetime, datetime, varchar(18), datetime을 자료형으로 가지게 된다. Payment entity에서 언급했던 것과 마찬가지로, logical model에서는 자료형을 DATETIME DAY TO MINUTE으로 설정했지만, physical model에서는 datetime으로 바뀐 것을 확인 할 수 있었다.

해당 entity에서는 actual_arrival_time_date를 제외한 나머지 attribute들은 null 값을 허용하지 않도록 하였다. Primary key이자 foreign key인 shipment_ID는 해당 값을 통해 tuple을 unique하게 식별함과 동시에 shipment entity와 무결성 체크를 하게 된다.
Actual_arrival_time_date를 제외한 나머지 attribute들은 모두 예상이나 어떤 시간대에 배송이 진행될지에 관한 정보를 담기 때문에 null을 허용하지 않고 어떤 값이라도 저장하는게 구현적으로 맞다고 판단되어 null을 허용하지 않도록 하였다.
Actual_arrival_time_date의 경우, 실제 배송이 완료된 시간을 저장할 것인데 아직 배송중이거나 사고 등으로 인해 도착이 지연될 수 있기 때문에 해당 부분은 null을 허용하도록 설정했다.

Shipment_ID는 number, 그 외 나머지는 string으로 domain이 설정되었다.

Shipment와 1:1 relationship을 형성한다. 하나의 shipment에 대해서 하나의 배송시간에 대한 정보를 가진 dataset이 존재할 것이기 때문이다.

xi. Location entity

Location entity는 location_ID를 primary key로 가지게 된다. 해당 attribute는 char(6)으로 선언되어 있으며 고정된 6자리의 string을 data로 받게 된다. 나머지 attribute들인 address와 location_name은 varchar(18)로 선언되어 있으면 최대 18글자까지의 data를 정보로 저장할 수 있다.

Location entity는 장소에 대한 정보를 저장하기 때문에 모든 attribute가 null을 허용하지 않도록 설정했다. Location_ID는 우편번호, location name과 address는 세부 주소를 저장하게 될 것인데 일반적으로 세 항목 모두 고유의 값을 가질 것이고 null인 경우가 아마 없을 것이기 때문에 null을 허용하지 않도록 설정했다.

해당 entity의 attribute들 모두 string으로 domain이 설정되었다.

Shipment_delivery와는 1:N의 relationship을 형성한다. 배송 정보를 조회하게 될 때, location의 경우 계속 바뀔 수 있기 때문에 하나의 배송 정보가 여러 location에 대응될 수 있어야 한다.

xii. Transportation entity

Transportation entity는 transportation ID, transportation_status를 primary key로 가진다. 해당 attribute들은 integer와 varchar(18)을 자료형으로 가지며 정수나 최대 18글자만의 정보를 저장할 수 있다. 나머지 attribute들은 type_of_transportation, capacity, shipment_ID가 있다. Type_of_transportation은 varchar(18)로 선언되어 있으며 최대 18글자만큼의 정보를 저장할 수 있다. Capacity는 integer형으로 선언되어 있고 transportation의 용량에 대한 정보를 저장할 것이다. Shipment_ID는 foreign key로 shipment entity를 referencing 하고 있다. 따라서, shipment에 존재하는 정보만이 해당 entity에서 tuple 값으로 들어오게 될 것이다.

해당 entity의 attribute들은 모두 null을 허용하지 않도록 설정했다. Transportation_ID와 transportation_status, shipment_id는 primary key에 해당되기 때문에 null을 포함할 수 없다. Type_of_transportation과 capacity의 경우, 해당 entity에 들어오는 transportation들은 배송을 진행하게 되고 Type_of_transportation과 capacity은 특정 shipment의 무게를 고려했을 때 필수적으로 기입되어 있어야 논리적으로 맞는 tuple을 entity에 집어 넣을 수 있을 것이므로 null을 허용하지 않도록 설정했다.

Transportation_status와 type_of_transportation은 string, 나머지는 number로 domain이 설정되었다.

Transportation_shipment relationship에서 하나의 shipment는 여러 개의

transportation 중 하나가 선택되어서 배송이 진행되고, transportation은 하나의 shipment를 배송한다고 가정한다. 따라서 해당 entity들 사이에 1:N의 relationship이 형성된다.

Shipment manager와의 relationship에서 transportation은 관리하는 다수의 관리자가 있을 것이고, manager 또한 여러 개의 transportation을 관리할 수 있을 것이다. 따라서, 두 entity 사이에 N:N의 relationship이 형성된다.

xiii. Manger entity

Manager entity는 manage_ID를 primary key로 가지게 된다. 해당 attribute는 integer형으로 선언되어 있다. 나머지 attribute로는 name, affiliation, phone_number 등이 존재하고 각각 varchar(18), varchar(18), char(11)을 자료형으로 가지게 된다.

해당 entity에서는 affiliation을 제외하면 null 값을 허용하지 않았다.Shipment manager는 물건을 담당하는 사람이기에 연락이가능해야 하고 primary key인 manager ID와 attribute name은 사람을 판별하는데 중요한 요소들이기 때문이다. Affiliation의경우, 소속이 없는 프리랜서일 가능성도 있을수 있어서 해당 부분은 null을 허용하도록 하였다.

Manager_ID는 number, 나머지는 string으로 domain이 설정되었다.

Transportation와의 relationship에서 transportation은 관리하는 다수의 관리자가 있을 것이고, manager 또한 여러 개의 transportation을 관리할 수 있을 것이다. 따라서, 두 entity 사이에 N:N의 relationship이 형성된다.

xiv. Customer_manager, manager_transportation

해당 entity들은 physical schema에서 entity들 사이에 있는 relationship이 many-to-many relationship임을 알리기 위해 logical schema diagram에서 physical schema diagram으로 변환되는 과정에서 erwin이 자동적으로 생성하게 되는 entity들이다.

B. ODBC implementation에 관한 설명

해당 부분은 visual studio에서 작성된 conn_test.cpp에서의 mysql workbench와의 연동 및 프로젝트 명세서에서 명시되어 있는 7가지 쿼리들에 대해 어떻게 처리했는지

기술하게 될 것이다.

i. MYSQL Workbench와의 연동

```
const char* host = "localhost";
const char* user = "root";
const char* pw = "1!2@3#4$";
const char* db = "project";
```

해당 파일의 header 부분 근처에 있는 이 값들을 기반으로 mysql workbench와 연동을 진행한다.

여기서 수정을 가하게 될 부분은 const char* pw와 const char* db 부분으로 사용자가 mysql workbench에서 생성한 DB schema의 이름을 const char* db에 저장하고 pw는 초기 설정을 진행하며 설정한 비밀번호로 설정하게 된다. 해당 부분은 본 사용자의 파일 이름과 pw이며, 제출을 하며 pw의 값이 'mysql'로 바뀌게 될 것이다.

ii. 프로그램 menu의 구현

```
void mainmenu() {
    system("cls");
    printf("------ SELECT QUERY TYPES ------₩n₩n");
    printf("       1. TYPE I₩n");
    printf("       2. TYPE II₩n");
    printf("       3. TYPE III₩n");
    printf("       4. TYPE IV₩n");
    printf("       5. TYPE V₩n");
    printf("       0. QUIT₩n");
    printf("       Choose : ");
}
```

Conn_test.cpp에서 메뉴의 구현 예시

위의 예시와 비슷하게 각 메뉴에 해당되는 부분들을 따로 함수로 선언해주었다. 명세서에 기재된 대로 초기 화면 (mainmenu에 해당)과 각 쿼리 타입에 맞는 메뉴를 printf() 함수를 통해 간단하면서도 직관적이게 프로그램 상에 출력하게 될 것이다.

메뉴를 구현함에 있어 system("cls")와 system("pause") 등을 사용했는데 이는 프로그램을 작동시키며 편의성과 가독성을 올리기 위한 것으로 각각 특정 상황에서 프로그램 상에 출력되어 있는 정보들을 모두 삭제시키거나 입력이 들어오기 전까지 프로그램의 화면을 멈추게 하는 역할을 한다. 해당 부분은, 작동되지 않을 가능성도 염두하여 해당 부분들을 주석처리한 파일도 같이 첨부할 것이다.

iii. CREATE TABLE & INSERT VALUES

```
CREATE TABLE customer (
    security_number CHAR(13),
    customer_name VARCHAR(18),
    phone_number CHAR(11),
    PRIMARY KEY (security_number)
);

insert into customer values ('8412251122541', 'Graham', '01254867745');
insert into customer_account values ('graham02', 'grahamtheruler', '8412251122541');
insert into payment values ('29103845', '3000', '2022-08-28 17:22:21', 'card', 'done', 'graham
```

MYSQL workbench 에서 작성한 table의 생성과 table에 tuple을 insert하는 코드 예시이다.

프로젝트 1과 2를 진행하며 생성한 logical 및 physical schema diagram을 기반으로 database에서 사용할 table을 위의 예시와 같게 create table {relation 이름} (attribute 정보...); 를 이용해서 생성한다. 본 레포트의 작성자는 프로젝트를 진행하며 13개의 entity를 얻었고 이에 따라 create table을 13번 수행하여 구현하고자 했던 database model에 맞게 table을 생성하였다. 해당 부분은, 코드를 붙여 넣으면 너무 길어짐으로 같이 첨부하게 될 DDL.sql 파일 혹은 create.txt 파일을 통해 확인할 수 있을 것이다.

Table의 생성이 완료되었다면 후술할 query들의 확인을 위해 table에 tuple을 넣을 필요가 있다. Tuple의 insert는 insert into {relation 이름} values (relation에서 선언된 attribute들에 대응되는 값들); 의 형태로 진행된다. Insert 문 또한 전부 다 붙여넣으면 너무 길어지므로 같이 첨부하게 될 smallRelationsInsertFile .sql이나 insert.txt 파일을 통해서 확인할 수 있을 것이다. 앞에서 create table문을 통해 생성한 table에 tuple을 insert하는 것으로 기본적인 준비는 끝나게 된다.

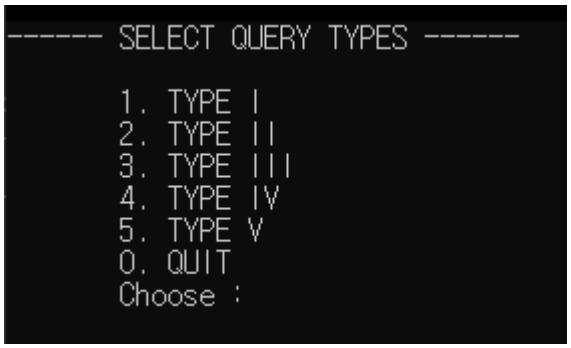
한 가지 주의사항은, create table문과 insert into 문 모두 foreign key constraint에 따라 선언 순서가 바뀌어야 한다는 것이다. 무결성 검사를 위해 table에 foreign key가 존재한다면, reference되는 table을 먼저 생성한 다음에 해당 foreign key를 가지는 table을 생성할 수 있다는 것이다.

Insert into문도 table과 마찬가지로 insert 순서를 foreign key constraint에 위배되지 않도록 진행해야 한다.

해당 부분은 file stream을 통해 해당 CRUD 문을 포함하는 .txt 파일의 내용

을 읽어 mysql_query()함수를 통해 mysql workbench와 연동하여 project schema에 의도한 CRUD 동작이 실행될 수 있도록 구현했다.

iv. Main menu 진입



Create와 insert를 통해 테이블 생성과 tuple insert를 마친 다음에 visual studio 상에서 conn_test.cpp 파일을 컴파일하여 실행하게 되면 다음과 같은 화면이 프로그램 상에 출력된다.

기본적으로, 해당 프로그램을 main menu에서 0의 입력이 들어올 때까지 while(1)을 통해 무한루프를 돌도록 하였다. 이는 같이 첨부하게 될 conn_text.cpp 파일에서 구현을 확인할 수 있을 것이다.

v. Main menu 이후의 진행

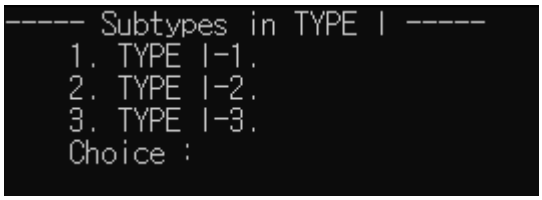
iv에서 언급한 것과 같이 main menu에서 0의 입력이 들어오기 전까지 프로그램은 계속 실행 될 것이다. 이후, 메뉴에서 출력되어 있는 번호에 해당되는 입력이 들어오게 되면, 해당 입력에 맞는 menu 화면이 출력될 것이다.

TYPE들의 menu에서는 0의 값이 들어오게 되면 이전의 menu 화면으로 돌아가게 될 것이다. TYPE II ~ TYPE V의 경우, 해당 메뉴에서 0을 입력으로 줄 경우, main 화면으로 돌아오게 될 것이다. (TYPE IV의 경우, 입력이 주어지지 않으므로 query 처리 결과를 보여 준 다음 main menu를 출력하게 될 것이다.)

TYPE I의 경우 TYPE의 메뉴에서 submenu의 선택을 추가적으로 해야한다. 만약 해당 화면에서 0을 입력으로 줄 경우 main menu로 돌아오게 된다. Submenu의 선택 후, submenu에 대한 화면이 출력된 다음 0을 입력으로 받게 되면 TYPE I의 출력화면으로 돌아오게 되도록 프로그램을 구성하였다.

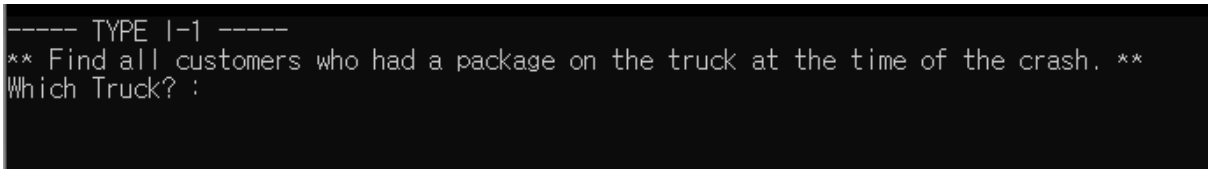
각 TYPE에서 실행될 select, from, where query문은 .cpp파일 내부에서 작성되었다.

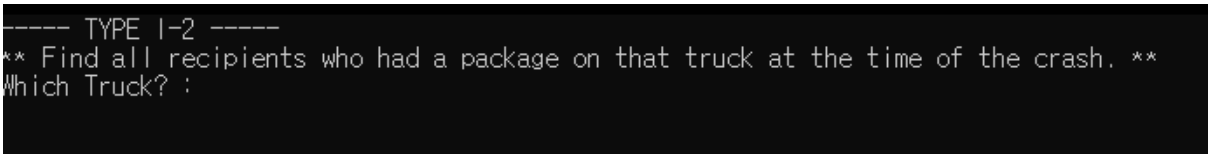
vi. TYPE I

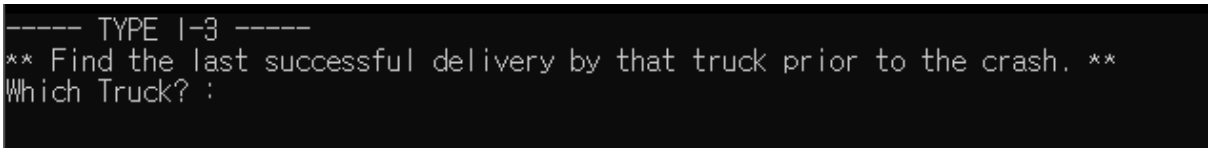


Main menu에서 1을 입력해서 TYPE I의 query 선택으로 들어갔을 때 나오게 되는 메뉴이다.

TYPE I에서는 3개의 submenu가 존재하며 이를 위해 각각 메뉴를 구현해주었다. 해당 submenu들에 진입했을 때의 프로그램 상의 출력은 다음과 같다.

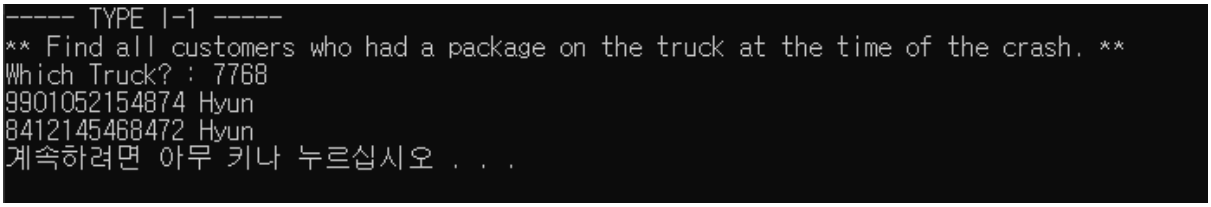


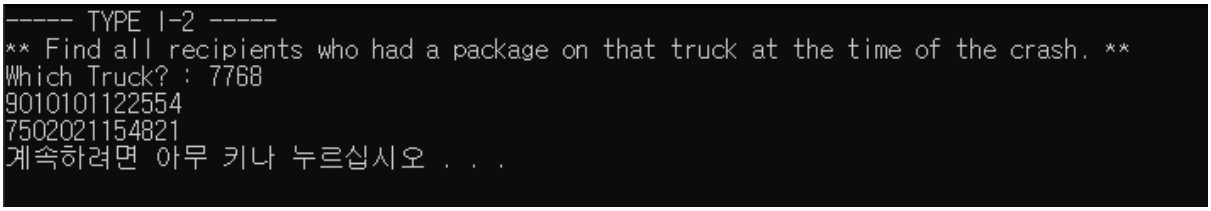


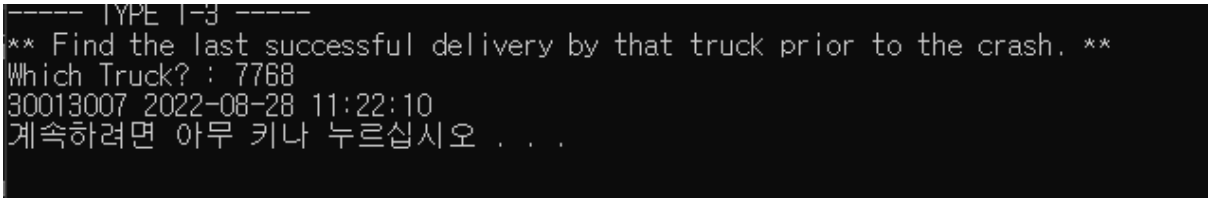


해당 쿼리의 input으로는 truck number가 주어진다고 명세서에 써져 있기 때문에 이를 기반으로 table과 tuple을 구성하게 되었다.

이 때, I-1와 I-2의 query는 at the time of the crash, 즉 사고가 일어났을 때 해당 truck에 package가 있었던 customer와 recipient에 대한 정보를 찾는 query이다. 하지만, 입력이 truck number이므로 언제 사고가 일어났는지를 입력으로 받지 않는다. 따라서, 해당 구현에서는 table에 tuple을 insert할 때, 사고가 이미 일어났다고 가정하여 transportation entity에서 transportation_status가 active, inactive, crash 상태 중 하나의 상태로 저장되도록 하였다.







TYPE I의 query들에 대한 실행결과이다. 동봉된 insert.txt 파일에서 crash와 관련된 transportation은 7768 하나 뿐이다.

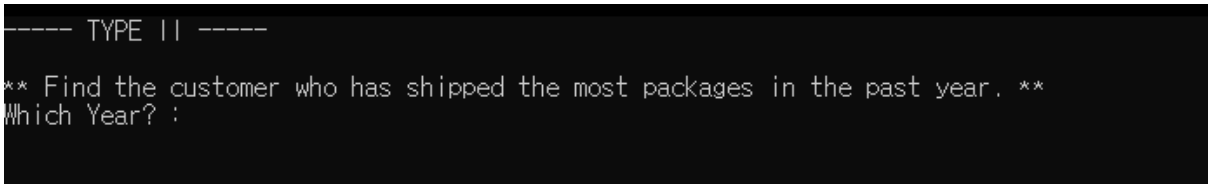
TYPE I-1의 query는 `select distinct c.security_number, c.customer_name from customer c right outer join package p on c.security_number = p.security_number right outer join shipment s on p.shipment_id = s.shipment_id where s.shipment_id in (select shipment_id from transportation t where transportation_id = '%d' and transportation_status = 'crash'` 로 구성되었다. 해당 query는 customer entity를 package, shipment와 join하여 구성된 테이블에서 status가 crash인 transportation에 의해 배송되던 package의 배송자의 정보가 출력되게 된다. 처리 결과는 customer의 security number, name 순으로 출력된다.

TYPE I-2의 query는 `select distinct r.recipient_id from recipient r right outer join package p on r.recipient_id = p.recipient_id right outer join shipment s on p.shipment_id = s.shipment_id where s.shipment_id in (select shipment_id from transportation t where transportation_id = '%d' and transportation_status = 'crash'` 로 구성되었다. 해당 query는 TYPE I-1에서의 query와 유사하다. Recipient entity를 package, shipment와 right outer join하여 현재 status가 crash인 transportation에 의해 배송되던 package의 수취인에 대한 간단한 정보가 프로그램 상에 출력된다. 처리 결과는 recipient의 recipient ID가 출력된다.

TYPE I-3의 query는 `select distinct ti. shipment_id, ti.actual_arrival_timedate from delivery d join transportation t on d.transportation_id = t.transportation_id join delivery_time ti on d.shipment_id = ti.shipment_id where (t.transportation_status != 'crash' and`

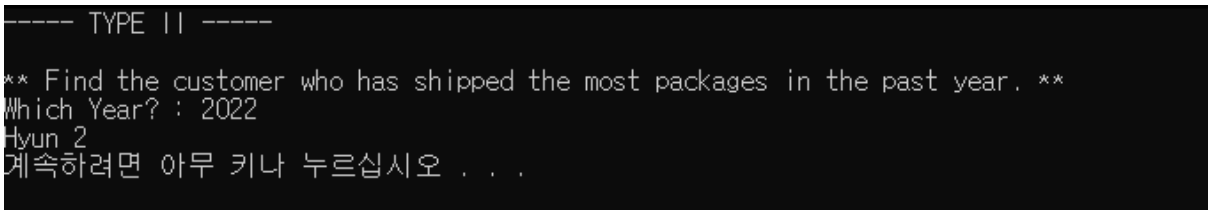
`t.transportation_id = '%d') order by ti.actual_arrival_timedate desc limit 1"` 로 구성되었다. Delivery entity를 delivery_time과 join한 table에서 transportation status가 crash가 아니었고 실제 배송 날짜 중에서 입력으로 받은 transportation이 진행한 배송 중, 가장 최근에 진행된 배송 하나를 프로그램 상에 출력하게 된다. 해당 query에서 배송 날짜를 기준으로 정렬하기 위해 order by ti.actual_arrival_timedate desc 를 사용했고 limit 1을 이용해 하나의 결과를 가지도록 하였다. 처리 결과는 마지막으로 배송 완료된 package의 package ID와 배송이 완료된 시간 순으로 출력된다.

vii. TYPE II



Main menu 에서 2번 입력을 통해 TYPE II의 query 처리를 위해 TYPE II의 menu를 불러온 화면의 캡처이다.

명세서에서 입력이 year로 진행된다고 명시되었으므로 입력으로 year의 값을 받게 된다. 동봉된 insert.txt 파일에서는 2022년에 일어났던 서비스에 대해서만 기록하고 있다.



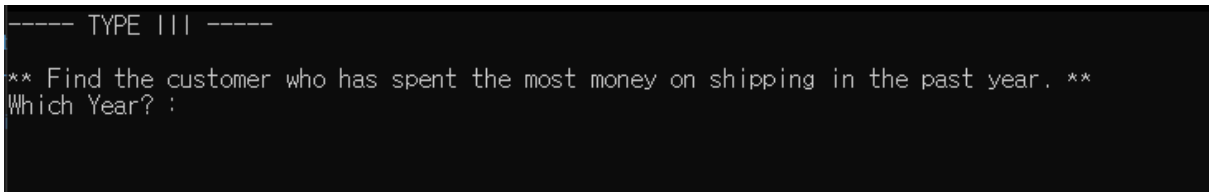
해당 query에 대한 처리 결과이다.

TYPE II의 sql query는 다음과 같다. `select c.customer_name, COUNT(*) as packagecount from customer c join package p on c.security_number = p.security_number join payment s on s.payment_id = p.payment_id where year(s.payment_date) = %d group by c.security_number order by packagecount desc limit 1`

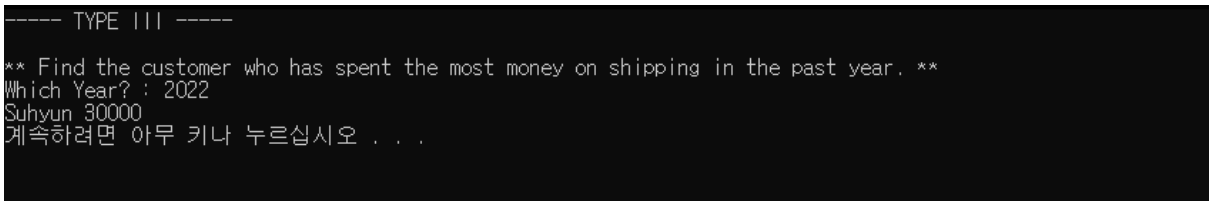
Customer entity를 package, payment와 join한 table에서 customer의 이름과 packagecount로 정의한 COUNT(*)에 대해 customer의 security_number로 내림차순 정렬한 결과에 대해 입력으로 주어진 year의 count가 가장 많았던 customer 한 사람을 화면 상에 출력하도록 하였다. 이 때 order by packagecount desc가 내림차순 부분에 해당되고 limit 1이 그 중, 제일 위에 있

던 값 하나 만을 출력할 수 있는 역할을 담당하게 된다. 처리 결과는 customer의 이름, 해당 customer가 보낸 package의 수가 출력된다.

viii. TYPE III



Main menu에서 3의 입력을 통해 TYPE III의 메뉴 화면으로 진행 했을 때 프로그램 상에 출력되는 결과이다.



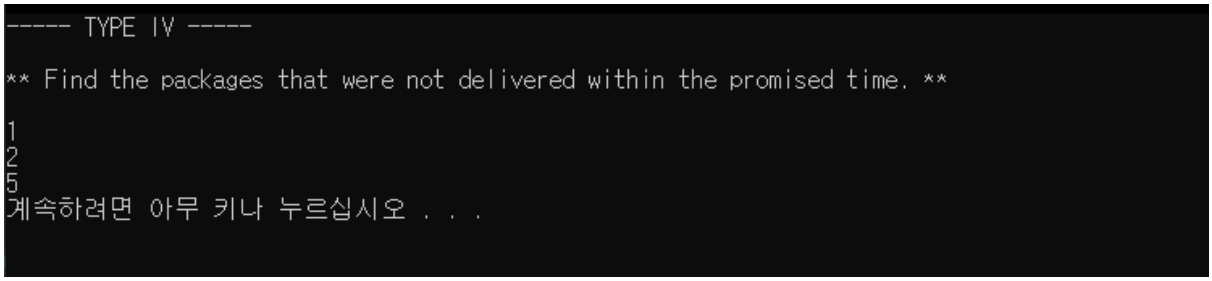
Table에 insert되어 있는 tuple의 정보를 기반으로 specific year을 입력으로 주었을 때의 실행 결과이다. 동봉된 insert.txt 파일에서는 2022년에 일어났던 서비스에 대해서만 기록하고 있다.

TYPE III의 sql query문은 다음과 같다. `select c.customer_name, sum(s.payment_amount) as total_payment from customer c join customer_account ac on c.security_number = ac.security_number join payment s on ac.account_id = s.account_id where year(s.payment_date) = %d group by c.security_number order by total_payment desc limit 1"`

여기서 %d의 부분에 입력으로 주어진 year의 값이 들어가게 된다.

해당 query는 customer entity를 customer_account, payment와 join한 table에서 특정 account의 payment_date 중 year에 해당되는 부분을 security number로 묶은 다음, payment_amount를 sum을 통해 구한 것을 내림차순 정렬하게 된다. 이 때, payment_amount의 sum값이 가장 컸던 한 사람을 프로그램 화면 상에 출력하는 query이다. 처리 결과는 해당 년도에 가장 많은 돈을 지불한 customer의 이름, 그 때의 액수 순으로 출력된다.

ix. TYPE IV



Main menu에서 4번 입력을 통해 TYPE IV의 query처리를 위해 TYPE IV의 메뉴로 진입 했을 때의 화면이다.

명세서에서 TYPE IV는 입력이 주어지지 않는다고 써져 있었으므로 해당 메뉴의 진입과 동시에 query문을 실행하여 해당 결과에 맞는 값들을 프로그램 화면 상에 출력되도록 하였다.

TYPE IV의 query는 다음과 같다.

```
select distinct package_id from package p join delivery_time t on
p.shipment_id = t.shipment_id join delivery d on p.shipment_id = d.shipment_id
where (d.transportation_status = 'crash' or t.actual_arrival_timedate >
t.estimated_delivery_timedate)
```

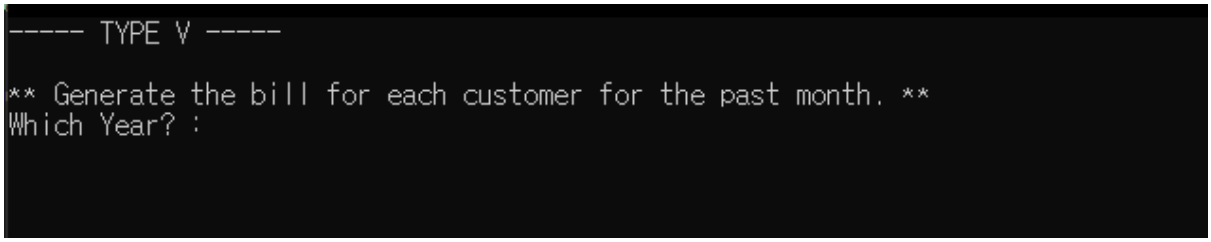
package entity를 delivery_time, delivery와 join을 한 table에서 actual_arrival_timedate가 estimated_delivery_timedate보다 크거나 transportation의 상태가 'crash'인 package들의 id를 화면 상에 출력하도록 하였다.

해당 조건에 부합하는 정보들만 출력되게 한 이유로는 기본적으로 customer는 estimated_delivery_timedate 이내에 배송이 완료되기를 원할 것인데 actual_arrival_timedate가 estimated_delivery_timedate보다 크다는 의미는 해당 배송이 예상 시간보다 늦게 완료되었다는 소리이다. 따라서, 4번 쿼리의 목적인 실제 배송 시간이 promised time과 달랐던 경우를 출력하는 것에 있어서 promised time보다 빠르게 배송이 진행된 것을 포함해서 출력하는 것보다 promised time보다 늦게 배송이 완료된 데이터를 뽑는 것이 해당 query의 목적에 조금 더 부합하다고 판단했다.

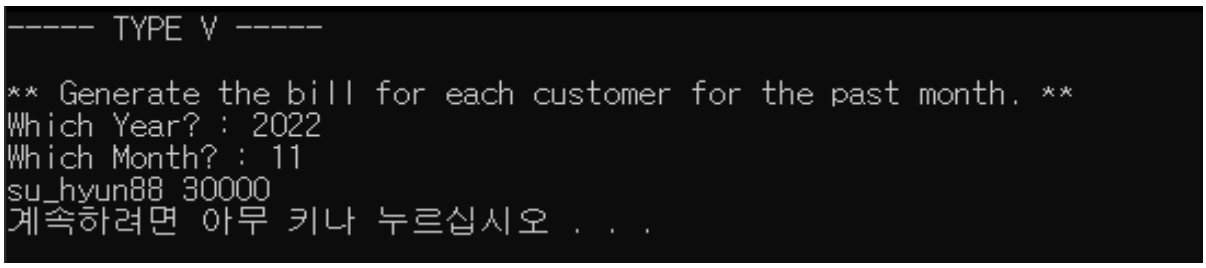
Transportat_status가 crash인 정보도 select하라고 한 이유는 transportation이 사고로 인해 crash된 상태라면 일반적으로 예상 시간 내에 배송이 완료되는 것을 기대하기는 어렵기 때문이다. 따라서, query의 수행 결과에 해당 조건에 부합되는 값이 있다면 같이 select 되도록 query 문을 작성하게 되었다.

처리 결과는 제 시간에 배송이 완료되지 못한 package의 package ID가 일렬로 출력된다.

x. TYPE V



Main menu에서 5번 입력을 통해 TYPE V의 처리를 위해 TYPE V의 menu로 진입했을 때의 화면이다.



본 프로젝트의 구현에서는 bill이 package의 payment를 진행하게 될 때 발생하는 모든 비용을 합친 값이 된다는 전제 하에 진행되었다. 동봉된 insert.txt 파일에서는 2022년에 일어났던 서비스에 대해서만 기록하고 있다.

입력으로 2개의 숫자가 주어지게 된다. 하나는 payment가 발생한 year, 나머지 하나는 해당 payment가 발생했던 month의 다음 month에 해당되는 값이다.

TYPE V의 sql query는 다음과 같다. `select p.account_id, sum(p.payment_amount) as bill from payment p where '%d' = year(p.payment_date) and '%d' = month(p.payment_date) group by p.account_id order by bill desc`

Payment entity에서 특정 account_id가 진행한 payment_amount의 값을 sum을 통해 구하게 된다. 이 때, sum을 통해 더하는 값은 `'%d' = year(p.payment_date) and '%d' = month(p.payment_date)` 이 부분에 의해 통제되게 된다. 입력으로 들어온 year, month 의 값이 같은 경우에만 sum에 값을 더해주는 것이다. 여기서 %d에 들어가는 month 값의 경우, query에서 past month에 대한 정보를 요구했기 때문에 실제로 입력으로 들어오게 되는 month의 이전 달에 대한 정보 중 query의 조건에 맞는 결과들을 출력하게 될 것이다.